### Redis和 mysql

（1）类型上

从类型上来说，mysql是关系型数据库，redis是缓存数据库

（2）作用上

mysql用于持久化的存储数据到硬盘，功能强大，但是速度较慢

redis用于存储使用较为频繁的数据到缓存中，读取速度快

（3）需求上

mysql和redis因为需求的不同，一般都是配合使用。

安装：

启动mongodb服务器：mongod

修改默认端口：mongod --port 新的端口号

mongodb默认的端口：27017

设置mongodb数据库的存储路径：mongod --dbpath 路径

⭐ 连接mongodb数据库：mongo

三大基本概念：

**数据库 database**

**集合(数组) collection**

类似与SQL中的数据表，本质上是一个数组，里面包含看多个文档对象，[{},{},{}]

**文档对象 document**

类似与SQL中的记录，一个文档对象{}就是一条记录

一个数据库由多个集合构成，一个集合包含多个文档对象。

**基本使用**：

show dbs 或show databases

查看所有的数据库

use xxx

切换到指定的数据库

db

查看当前操作的数据库

show collections

查看当前数据库中所有的集合

**数据库的CRUD操作:**

### 插入数据

插入一条数据

db.collectionName.insertOne( {name:'liu'} )

db表示的是当前操作的数据库

collectionName表示操作的集合，若没有，则会自动创建

插入的文档如果没有手动提供\_id属性，则会自动创建一个

插入多条数据

db.collectionName.insertMany( [ {name:'liu5'} , {name:'liu6'} ] )

需要用数组包起来

万能API：db.collectionName.insert()

#添加两万条数据

for(var i=0;i<20000;i++){

db.users.insert({username:'liu'+i}) #需要执行20000次数据库的添加操作

}

db.users.find().count()//20000

#优化：

var arr=[];

for(var i=0;i<20000;i++){

arr.push({username:'liu'+i})

}

db.user.insert(arr) #只需执行1次数据库的添加操作，可以节约很多时间

### 查询数据

db.collectionName.find() 或db.collectionName.find({})

查询集合所有的文档，即所有的数据。

查询到的是整个数组对象。在最外层是有一个对象包裹起来的。

db.collectionName.count()或db.collectionName.length() 统计文档个数

db.collectionName.find({\_id:222})

条件查询。注意：结果返回的是一个数组

db.collectionName.findOne() 返回的是查询到的对象数组中的第一个对象

注意：

> db.students.find({\_id:222}).name //错误

> db.students.findOne({\_id:222}).name //正确

# 1.mongodb支持直接通过内嵌文档的属性值进行查询

# 什么是内嵌文档：hobby就属于内嵌文档

{

name:'liu',

hobby:{

movies:['movie1','movie2'],

cities:['zhuhai','chengdu']

}}

db.users.find({hobby.movies:'movie1'}) //错误

db.users.find({"hobby.movies":'movie1'})//此时查询的属性名必须加上引号

#2.查询操作符的使用

#比较操作符

$gt 大于

$gte 大于等于

$lt 小于

$lte 小于等于

$ne 不等于

$eq 等于的另一种写法

db.users.find({num:{$gt:200}}) #大于200

db.users.find({num:{$gt:200,$lt:300}}) #大于200小于300

$or 或者

db.users.find(

{

$or:[

{num:{$gt:300}},

{num:{$lt:200}}

]

}

) #大于300或小于200

#3.分页查询db.users.find().skip(页码-1 \* 每页显示的条数).limit(每页显示的条数)

db.users.find().limit(10) #前10条数据

db.users.find().skip(50).limit(10) #跳过前50条数据，即查询的是第61-70条数据，即第6页的数据

#4.排序db.emp.find().sort({sal:1})

#1表示升序排列，-1表示降序排列db.emp.find().sort({sal:1,empno:-1})

#先按照sal升序排列，如果遇到相同的sal，则按empno降序排列

#注意：**skip,limit,sort可以以任意的顺序调用，最终的结果都是先调sort，再调skip，最后调limit**

#5.设置查询结果的投影，即只过滤出自己想要的字段

db.emp.find({},{ename:1,\_id:0}) #在匹配到的文档中只显示ename字段

### 修改数据

# 1.替换整个文档# db.collectionName.update(condiction,newDocument)

> db.students.update({\_id:'222'},{name:'kang'})

# 2.修改对应的属性，需要用到修改操作符，比如$set,$unset,$push,$addToSet

db.collectionName.update(

# 查询条件

{\_id:222},

{

#修改对应的属性

$set:{

name:'kang2',

age:21

}

#删除对应的属性

$unset:{

gender:1 //这里的1可以随便改为其他的值，无影响

}

}

)

# 3.update默认与updateOne()等效，即对于匹配到的文档只更改其中的第一个# updateMany()可以用来更改匹配到的所有文档

db.students.updateMany(

{name:'liu'},

{

$set:{

age:21,

gender:222

}

}

)

# 4.向数组中添加数据

db.users.update({username:'liu'},{$push:{"hobby.movies":'movie4'}})

#如果数据已经存在，则不会添加

db.users.update({username:'liu'},{$addToSet:{"hobby.movies":'movie4'}})

# 5.自增自减操作符$inc

{$inc:{num:100}} #让num自增100

{$inc:{num:-100}} #让num自减100

db.emp.updateMany({sal:{$lt:1000}},{$inc:{sal:400}})

#给工资低于1000的员工增加400的工资

### 删除数据

# 1. db.collectionName.remove()

# remove默认会删除所有匹配的文档。相当于deleteMany()

# remove可以加第二个参数，表示只删除匹配到的第一个文档。此时相当于deleteOne()

db.students.remove({name:'liu',true})

# 2. db.collectionName.deleteOne()

# 3. db.collectionName.deleteMany()

db.students.deleteOne({name:'liu'})

# 4. 删除所有数据：db.students.remove({})----性格较差，内部是在一条一条的删除文档。

# 可直接通过db.students.drop()删除整个集合来提高效率。

# 5.删除集合db.collection.drop()

# 6.删除数据库db.dropDatabase()

# 7.注意：删除某一个文档的属性，应该用update。remove以及delete系列删除的是整个文档

# 8.当删除的条件为**内嵌的属性**时：

db.users.remove({"hobby.movies":'movie3'})

### 文档之间的关系：

一对一

一对多

#用户与订单：

db.users.insert([

{\_id:100,username:'liu1'},

{\_id:101,username:'liu2'}

])

db.orders.insert([

{list:['apple','banana'],user\_id:100},

{list:['apple','banana2'],user\_id:100},

{list:['apple'],user\_id:101}

])

查询liu1的所有订单：

首先获取liu1的id: var user\_id=db.users.findOne({name:'liu1'}).\_id;

根据id从订单集合中查询对应的订单： db.orders.find({user\_id:user\_id})

多对多

#老师与学生

db.teachers.insert([

{

\_id:100,

name:'liu1'

},

{

\_id:101,

name:'liu2'

},

{

\_id:102,

name:'liu3'

}

])

db.students.insert([

{

\_id:1000,

name:'xiao',

tech\_ids:[100,101]

},

{

\_id:1001,

name:'xiao2',

tech\_ids:[102]

}

])

### mongoose:

**简介：**

1.mongoose是nodejs中的专门用于操作mongodb数据库的**js库**

2.mongoose中的对象：

Schema 模式对象（用于约束文档的结构）

Model 模型对象（即mongodb中的集合）

Document 文档对象（即mongodb中的文档）

**安装：**

npm i -s mongoose

连接数据库：

// 1.引入mongooseconst mongooes = require("mongoose");

// 2.连接mongodb数据库（url）

mongooes.connect("mongodb://localhost/users", {

useNewUrlParser: true,

useUnifiedTopology: true,});

// 3.监听mongodb数据库的连接状态

// 绑定数据库连接成功事件

mongooes.connection.once("open", function () {

console.log("连接成功");});

// 绑定数据库连接失败事件

mongooes.connection.once("close", function () {

console.log("数据库连接已经断开");});

// 4.断开数据库连接(一般不用)

mongooes.disconnect();

创建模式对象和模型对象：

//使用mongooes.schema创建模式，

const Schema=mongooes.schema;

//创建模式对象

//使用模式new模式对象

const stuSchema=new Schema({

name:String,

age:Number,

gender:{

type:String,

default:'female'

},

address:String})

//创建模型对象

//使用模式对象.model创建（集合，模式对象）

const StuModel=stuSchema.model("student",stuSchema);

//第一个参数表示创建的集合的名称，第二个参数表示利用的模式对象

利用模型对象进行增删查改操作：

添加操作：

UserModel.create({ user\_id: 100, name: "liu1" }, function (err) {

if (!err) {

console.log("插入成功");

} else {

console.log(err);

}});

let data = [

{ user\_id: 101, name: "liu2", age: 22 },

{ user\_id: 102, name: "liu3" },];UserModel.create(data, function (err) {

console.log(arguments[1]);

//第二个值表示的是所添加的文档对象,是一个数组});

查询操作：

/\* 查询: model.find(conditions,[projection],[options],callback)

conditions:查询的条件

projection:投影 { name: 1, gender: 1, \_id: 0 } 或 'name gender -\_id'

options:查询选项 { skip: xx, limit: xx }

model.findOne(...)

model.findById(...)

model.countDocuments(conditions,callback) 查询文档的数量 \*/

UserModel.find({}, function (err, data) {

console.log(data);});

UserModel.find(

{ name: /liu/i },

"name gender -\_id",

{ skip: 2, limit: 1 },

function (err, data) {

console.log(data); //返回的是一个文档对象数组

});

UserModel.findById("5f9fbfba14319e492c0f5bc4", function (err, data) {

console.log(data);

console.log(data instanceof UserModel);

//true 返回的文档对象属于模型对象（即集合）的实例对象});

UserModel.countDocuments({}, function (err, data) {

console.log(data);});

修改操作：

/\* 修改： model.update(conditions,[doc],[options],callback)

doc:修改后的文档对象

model.updateMany(...)

model.uodateOne(...)\*/

UserModel.updateOne({ name: "liu1" }, { $set: { age: 22 } }, function (err,data) {

if (!err) {

console.log("修改成功");

}});

UserModel.find({ name: "liu1" }, function (err, data) {

console.log(data);});

删除操作：

/\* 删除：model.remove(conditions,callback)

model.deleteOne(...)

model.deleteMany(...)\*/

UserModel.remove(

{

name: "liu2",

},

function (err, data) {

console.log("删除成功");

});

UserModel.find({}, function (err, data) {

console.log(data);});

**模块化处理：**

1. 单独创建一个数据库连接文件dbconncet.js

const mongooes =require("mongoose");

mongooes.connect("mongodb://localhost/mongooes\_test", {

useNewUrlParser: true,

useUnifiedTopology: true,});

mongooes.connection.once("open", function () {

console.log("连接成功");});

2.为每一个集合创建一个模型对象文件xxxModel.js

const mongooes = require("mongoose");

const Schema = mongooes.Schema;

const userSchema = new Schema({

user\_id: String,

name: String,

age: Number,

gender: {

type: Number,

default: 0,

},});

const UserModel = mongooes.model("user", userSchema);

module.exports = UserModel;

3.在最终的文件index.js中引入数据库连接文件和创建模型的文件：

const mongooes = require("./dbconncet");

const PostModel = require("./models/postModel");

PostModel.findOne({}, function (err, data) {

if (!err) {

console.log(data);

}});